Livrable 3.2.1	
Version	**1.1**
Date	**10/03/16**
Auteur(s)	francetélévisions
N° du Lot	**3.2**

Le lecteur media accessible à tous

Livrable 3.2.1 : Rapport d'étude et ses conséquences sur la conduite du sous-projet

Titre du projet	media4Dplayer
Abréviation	M4DP
Désignation	media4Dplayer, le lecteur media accessible à tous.
Durée du projet	De Janvier 2015 à Juin 2016 – 18 mois
Coordinateur projet	France Télévisions
Partenaires du lot	FRANCE TELEVISIONS (FTV)
Sous-traitant du lot	IRCAM
Organisme labellisateur	CAP DIGITAL
Financeurs	La Région Ile-de-France La BPIfrance
Titre de subvention	Fonds Unique Interministériel – FUI18

Le projet media4Dplayer, lecteur media accessible à tous.

Rapport d'étude et ses conséquences sur la conduite du sous-projet

Date de soumission : 10/03/16
Version : 1.1

Objectif(s) du livrable

Étudier les API Web Audio, les interfaces multimédia HTML5, l'interopérabilité entre les différents formats de flux et de fichiers, la compatibilité avec les navigateurs mobiles et desktop, la faisabilité d'une synchronisation "fine" second écran avec ces interfaces.

Historique	Date	Modification(s)
V 1.0	15/03/15	Création du document
V 1.1	10/03/16	Mise à jour mineure

Le projet media4Dplayer

Media4Dplayer est un projet collaboratif labellisé par le pôle de compétitivité Cap Digital et subventionné au titre du Fonds Unique Interministériel (FUI) par la région Île de France et BPIfrance. Ce projet de recherche et de développement s'inscrit dans la stratégie de Cap Digital, autour des thématiques d'accessibilité des contenus, de développement numérique et de Silver économie.

Durée de projet 18 mois : Janvier 2015 – Juin 2016

Avertissement

Les informations contenues dans ce document peuvent être sujet à modification sans préavis. Société ou noms de produits mentionnés dans ce document peuvent être des marques ou des marques déposées de leurs sociétés respectives.

Tous les droits sont réservés

Table Des Matières

1. Introduction

france**télévisions** a lancé une consultation dans le but d'adresser les problématiques audio et d'interopérabilité entre les différents navigateurs HTML5, l'API Web Audio et l'application media4Dplayer. Les réponses des différents candidats ont permis de statuer sur les processus audio avancés qu'il était possible de développer dans le temps et le budget impartis, en visant la finalité d'une preuve de concept et non d'un produit commercial. Dans les solutions que proposaient les candidats, certaine étaient clairement axées sur les performances et pointaient du doigt des mises à jour profondes à prévoir au niveau du moteur audio des navigateurs, afin de garantir un respect des latences audio/vidéo et répondre strictement au cahier des charges fixé par france**télévisions**. Une autre approche, plus légère et clé en main décrivait des développements centrés sur l'utilisation des fonctionnalités strictement offertes par l'API Web Audio. C'est cette piste, proposée par l'**IRCAM**, qui fut finalement retenue. Les développements ont pu démarrer dans la foulée, selon le principe de la méthode agile.

2. L'API Web Audio

Standardisée par le W3C, l'API Web Audio est un moteur de rendu audio codé en langage JavaScript qui offre un système puissant et flexible pour contrôler des données audio sur internet. Elle permet aux développeurs de choisir des sources audio (microphone, flux, média), d'y ajouter des effets, de créer des visualisations, d'appliquer des effets de spatialisation et bien plus encore. Les dernières spécifications sont proposées à cette URL : https://www.w3.org/TR/webaudio/

Même si les navigateurs web s'appuient sur les définitions du W3C pour offrir de nouvelles fonctionnalités, ils n'intègrent pas toujours les dernières normes et restent potentiellement incompatibles avec certaines spécifications. Quelques sites web recensent les navigateurs compatibles avec l'API Web Audio, en fonction des différentes versions logicielles et des plates-formes : http://caniuse.com/ - feat=audio-api ...autre exemple pour le support 3D avec WebGL: http://caniuse.com/ - search=webgl

Les fonctionnalités proposées par l'API Web Audio doivent donc être implémentées dans les navigateurs eux-mêmes, c'est à dire dans leur langage natif, C et/ou C++. Dans une couche plus haute, l'API Web Audio permet de piloter ces processus directement en JavaScript et indépendamment du navigateur, de l'OS et du hardware. Derrière cette API commune, le traitement va s'effectuer dans le code défini par le navigateur.

Dans le cas de Chromium/Google Chrome, le navigateur de référence retenu dans le cadre du projet, c'est la librairie Blink qui implémente les fonctionnalités offertes par l'API Web Audio (http://www.chromium.org/blink). Il peut être important de noter que chaque navigateur possède sa propre bibliothèque pour supporter les API, ce qui peut fournir des résultats potentiellement différents en fonction des implémentations (liés à la précision de calcul, à la définition du traitement exact, etc.). Par exemple, on constate des écarts de rendu entre les navigateurs Chrome et Firefox, alors qu'on sollicite le même processeur de dynamique Web Audio avec des réglages identiques.

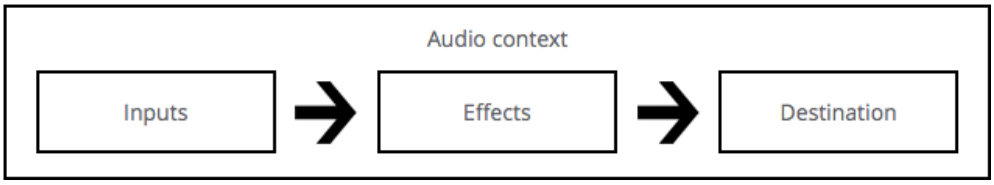
Malgré ces considérations techniques, l'API Web Audio offre un environnement de développement flexible et multi-supports, c'est un prérequis essentiel pour une application finale à destination du grand public. Les fonctionnalités offertes permettent de se projeter sur des usages dans lesquels l'audio est un vecteur d'accessibilité, depuis la personnalisation de l'écoute pour des malvoyants, jusqu'à la proposition d'une expérience utilisateur optimale pour tout à chacun. L'intégration avec l'application media4Dplayer est assez triviale puisque le lecteur Dash.js, retenu dans le cadre du projet, pilote directement le moteur audio du navigateur. Il décode le format MPEG-DASH présenté à l'entrée, l'API Web Audio joue ensuite uniquement son rôle de moteur audio : elle capture les buffers d'entrée, les traite, puis les retourne à l'application cliente.

2.1. Concepts et usages

L'API Web Audio implique le traitement d'opérations dans un **environnement audio**, elle a été conçue pour supporter le **routing modulaire**. Les opérations audio basiques sont réalisées grâce à des **nœuds audio,** reliés entre eux pour former un **graphe de routage audio**. Plusieurs sources avec différents types d'agencements de canaux sont supportés (jusqu'à 32 canaux), le tout dans un seul environnement. Ce design modulaire et flexible permet de créer des fonctions audio complexes avec des effets dynamiques.

Les nœuds audio sont reliés à travers leurs entrées et sorties, ils forment une chaîne qui commence avec une ou plusieurs sources et passe par un ou plusieurs nœuds avant d'atteindre une destination (nul besoin de préciser la destination si l'on souhaite uniquement visualiser le flux audio). Un processus de développement classique avec l'API Web Audio ressemble à ceci :

1. Création d'un environnement audio ;
2. Dans cet environnement, création des sources tel qu'un élément HTML5 *<audio>*, un oscillateur, un flux ;
3. Création de nœuds d'effets comme la réverbération, les filtres biquad, la balance, la compression de dynamique ;
4. Choix de la sortie audio, par exemple le casque d'écoute en mode stéréophonique ou une interface audionumérique alimentée en canaux discrets ;
5. Connexion des sources aux effets, et des effets à la destination.



La gestion du temps est contrôlée avec une grande précision et une latence la plus faible possible (dépendante du système, voir chapitre 3), ce qui permet aux développeurs d'écrire un code qui réagit précisément aux événements. Ce code sera aussi capable de s'adresser à des échantillons précis, même avec une fréquence d'échantillonnage élevée (jusqu'à 96 kHz). Sont ainsi envisageables des applications telles que des boîtes à rythme ou des séquenceurs.

L'API Web Audio permet également de contrôler la spatialisation du son selon différents modes de *panning*. En utilisant un système basé sur le modèle *source-listener,* elle autorise le contrôle du type de balance et la gestion de l'atténuation du son en fonction de la distance (avec effet doppler), induite par un déplacement de la source sonore ou de l'auditeur.

2.2. Interfaces de l'API Web Audio

L'API Web Audio possède au total 28 interfaces avec des événements associés, classés selon leur fonction en 9 catégories :

1. Définition du graphe audio : conteneurs et définitions qui donnent sa forme au graphe audio ;
2. Définition des sources audio : oscillateur embarqué, buffer audio, source audio composée d'un élément HTML5, source audio composée d'un *WebRTC MediaStream* (tel qu'une webcam ou un microphone) ;
3. Définition des filtres d'effets audio : gain, délai, processeur de dynamique, filtre biquad, convolution, agitateur d'harmoniques, forme d'onde périodique pour alimenter l'oscillateur embarqué ;
4. Définition des sorties audio ;
5. Analyse et visualisation des données : nœud d'analyse en temps réel, fréquentiel et temporel;
6. Séparation et réconciliation des canaux audio : *split* d'un flux multicanal en un flux monophonique, *merge* de canaux discrets en un flux multicanal ;
7. Spatialisation audio : réglage du point d'écoute (position et orientation), mode de *panning* des sources avec différents algorithmes (*"Equal Power"* ou *"HRTF"*) ;
8. Traitement audio JavaScript (*ScriptProcessorNode*) : en voie d'abandon, cette catégorie permet de créer des nœuds JavaScript spécifiques pour initier un nouveau processus audio. Exemple : les nœuds *"binauralFIR"* et *"binauralModeled"* codés par l'IRCAM, en vue d'améliorer la qualité du rendu binaural embarqué dans un environnement Web Audio. Une publication du 29/08/2014 témoigne du remplacement imminent de cette catégorie par une nouvelle, baptisée *Audio workers*. Hélas, cette dernière n'est pas encore implémentée dans les versions stables des navigateurs et seuls les "anciennes" interfaces ont pu être mise en œuvre ;
9. Traitement audio en tâche de fond : trois interfaces distinctes permettent de rendre un graphe audio très rapidement en mode "offline", poussant les données audio vers un nouveau buffer plutôt qu'une sortie physique ;
10. *Audio Workers* : cette nouvelle couche, en cours d'implémentation, offre la possibilité d'écrire des scripts de traitements audio directement sous la forme de plugins web (voir point n°8 précédent).

2.3. Ajout de fonctions supplémentaires dans l'API Web Audio

Plusieurs approchent cohabitent quant au développement de modules dans l'API Web Audio. Certains développeurs considèrent comme essentiels des critères comme la pérennité du code source et les performances induites. La figure 1 représente l'architecture globale d'un navigateur, elle est composée de différentes bibliothèques permettant de supporter les standards comme Web Audio.

Ces bibliothèques sont codées en langage natif (C/C++). Tel que défini dans l'API Web Audio, un nœud laisse au développeur la définition du traitement à effectuer. Même si les *ScriptProcessorNode* offrent une possibilité pour définir des nœuds en langage JavaScript, nous sommes actuellement dans une phase de transition car cette ancienne catégorie devrait céder la place à une nouvelle (les *Audio Workers*). Ses interfaces remplaceraient a terme la capacité offerte par l'API pour surcharger les traitements.

Pour faciliter la compréhension du point de vue de l'architecture, les nœuds qui sont un regroupement de nœuds élémentaires en vue de créer un process plus complexe peuvent être dissociés, et répondre à la dénomination de ***MetaNode***. À la suite du projet de recherche, ce sont ces types de nœuds qu'il conviendrait de mettre en œuvre pour répondre aux enjeux d'une industrialisation, afin de garantir les performances globales et indépendamment des multiples plates-formes clientes potentielles.

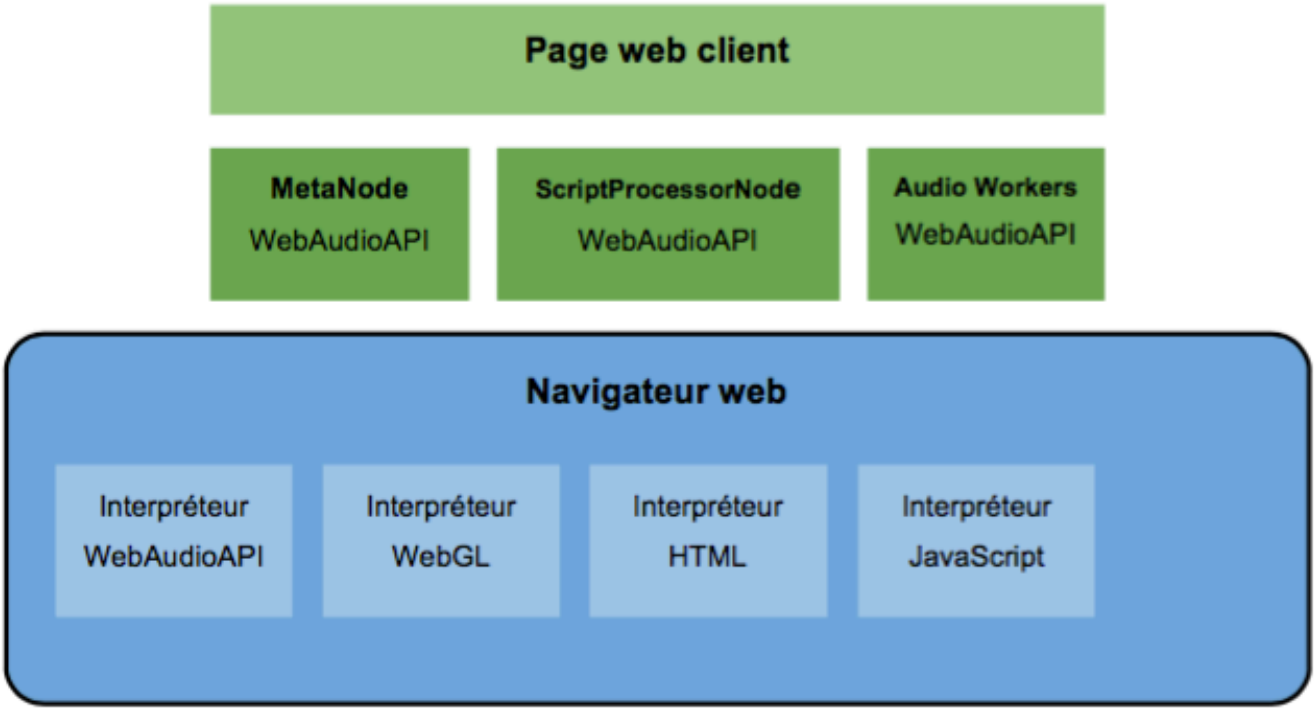


Figure 1 : architecture du navigateur web et des intégrations multi-niveaux

Voici un tableau comparatif des différentes possibilités de programmation, il propose la synthèse des avantages/inconvénients qui découlent d'un choix de type de codage :

	Programmation native	ScriptProcessorNodes / Audio Workers	MetaNodes
Langage	C/C++	JavaScript	JavaScript
Performances	+++	+	+++
Agilité	+	++	+++
Complexité	+++	++	+
Pérennité du code	+++ (si contribution à Web Audio API)	+ (problème du changement d'interface en cours)	++ (tant que l'API ne change pas)
Regroupement des traitements	+		+++
Portabilité	+	+++	+++

Le tableau précédent illustre le fait que chaque type de développement présente des avantages et des inconvénients. Contribuer directement à Web Audio API -c'est à dire développer en C/C++ à partir du code source du navigateur- est synonyme du maintien des performances, tout en préservant la pérennité du code. Mais cette démarche demande aussi plus d'efforts en matière d'intégration car elle implique des échanges réguliers avec les contributeurs officiels pour faire valider les modifications dans les couches basses. En attendant cette validation, il peut être envisageable de compiler une version personnalisée de Chromium qui intègre quelques développements spécifiques.

Les *MetaNodes* sont, quant à eux, assez pratiques pour concaténer des éléments et n'engendrent presque aucune perte de performances. Cette piste était techniquement la plus appropriée pour développer les modules décrits dans le cahier des charges de media4Dplayer, du moins lorsque les traitements requis ne nécessitent pas plus qu'un regroupement de nœuds élémentaires.

Au cours du projet, la communauté Web Audio n'a pas donné suite à la publication de ces nouvelles couches et les développeurs de l'IRCAM ont donc utilisé les nœuds natifs embarqués pour tous les traitements les plus simples, ainsi que les nœuds *ScriptProcessorNode* pour insérer des fonctionnalités supplémentaires. En aval du projet de recherche et accompagnés d'acteurs et des consortium de validation (EBU, W3C, SMPTE…), la piste de quelques développements C/C++ pour "patcher" les navigateurs web pourrait être poussée auprès des mastodontes Apple et Google.

3. Faisabilité d'une synchronisation "fine" second écran dans l'environnement HTML5.

Sur les bases d'un démonstrateur iOS natif, développé par un prestataire en 2011 et baptisé **FTVSync**, france**télévisions** a confié à l'IRCAM l'étude de l'intégration d'une telle solution de synchronisation dans l'environnement HTML5. Sur un second écran, la preuve de concept initiale permettait de présenter synchrones à la diffusion linéaire ou la catch-up plusieurs flux audio additionnels, dont l'audiodescription ou la version originale du programme. Au delà de la faisabilité du portage des algorithmes originels en langage JavaScript, l'étude de la question des latences induite par le système complet (système d'exploitation, moteur audio, couches logicielles annexes) reste une problématique majeure. Pour rendre le service souhaité, une lecture d'audio synchronisé avec quelques millisecondes maximum de décalage (qui garantit l'absence d'écho lors de la lecture), ces latences doivent être impérativement être connues, remontées par le système, et fixes.

En l'état actuel du standard HTML5, dont l'API Web Audio fait partie, il n'y a hélas pas d'accès aux latences des appareils, c'est un prérequis technique élémentaire pour connaître le moment exact auquel un son sera effectivement émis. Néanmoins, ce problème est en cours de discussion au sein du groupe de travail Web Audio : https://github.com/WebAudio/web-audio-api/issues/12. Une première implémentation est d'ores et déjà disponible pour Windows. Il s'agit de Crosswalks, qui permet d'embarquer un navigateur pour réaliser des applications multi plates-formes : https://github.com/crosswalk-project/chromium-crosswalk/pull/332

Le *Community Group* du W3C, *"Multi-device Timing for Web"*, travaille sur la question de la synchronisation entre différents appareils : https://www.w3.org/community/webtiming/. Mentionnons aussi un groupe plus spécifique aux cas d'utilisation liés à la télévision : https://www.w3.org/community/tvapi/. Au dessus des listes de discussions liées à ces groupes, François Daoust <fd@w3.org>, qui travaille pour le W3C, semble le contact le plus pertinent pour faire remonter les cas d'utilisation et pousser la standardisation.

Aujourd'hui, il est déjà possible de synchroniser les horloges de différents appareils :

- soit en utilisant un serveur spécifique et l'horloge *performance.now*, une interface présente dans le framework WebRTC, ce qui exclut néanmoins le navigateur Safari d'Apple : https://www.w3.org/community/webtiming/
- soit en faisant jouer uniquement les horloges audio (https://github.com/collective-soundworks/sync), ce qui nécessite quelques secondes à quelques minutes de mesure pour obtenir une précision de l'ordre de 1 à 10 ms, en fonction suivant des appareils. Avec des horloges synchronisées, il est possible ensuite de travailler sur une horloge de référence, ce qui permet de compenser en partie les variabilités liées aux communications et aux temps de traitement.

Néanmoins, même avec des horloges audio synchronisées, il est impossible d'assurer un rendu cohérent sans avoir connaissance de **la latence induite par le navigateur, le système d'exploitation lui-même, et le matériel utilisé**. La variabilité mesurée entre les systèmes est de l'ordre d'une demi-seconde. En attendant que l'information de latence soit présente dans le standard, il n'est pas d'autre choix que de la mesurer pour être à même de la compenser.

Pour automatiser cette mesure, une solution consiste à effectuer une mesure de latence entre l'entrée et la sortie de la chaîne complète, en admettant que le délai soit le même en entrée et en sortie. Hélas, l'accès au périphérique d'entrée audio ne fait pas partie du standard Web Audio. Il est effectué à l'aide de la fonction *getUserMedia*, embarquée dans WebRTC, qui n'est pas implémenté par Apple et donc indisponible pour tous les navigateurs sous iOS, dont Safari.

Une solution, mesurant la latence relative entre 2 appareils, permet d'effectuer une calibration par rapport à une référence (un appareil déjà calibré) : https://github.com/collective-soundworks/calibration/. Dans tous les cas, le processus est fastidieux, vu le nombre de systèmes et d'innombrables mises à jours périodiques, ce qui explique qu'il n'existe pas de base de données disponible. De plus, le véritable problème actuellement concerne la grande latence des systèmes Android : jusqu'à près d'une seconde pour les systèmes les moins performants !

Pour plus de détails sur ces différents mécanismes, un article en anglais de Lambert et al. *"Synchronization for Distributed Audio Rendering over Heterogeneous Devices in HTML5"*, mis à jour l'occasion de la **W**eb **A**udio **C**onference 2016 est disponible à l'adresse suivante :

https://smartech.gatech.edu/bitstream/handle/1853/54598/WAC2016-81.pdf

Conclusion

…sur les latences induites par les processus audio :

Lorsqu'on travaille l'audio dans un environnement JavaScript, il apparaît difficile d'obtenir un rendu sans faille ni "clip" à l'écoute, tout en conservant une faible latence (surtout lorsque le processeur est fortement sollicité).

La couche JavaScript reste beaucoup plus lente que du code C++ optimisé et ne bénéficie pas des avantages qu'apportent les optimisations SSE et multithreading des processeurs actuels. Un code natif optimisé peut s'avérer 20 fois plus rapide que du JavaScript pour certains calculs. La situation s'aggrave encore avec des traitements plus lourds, tels que la convolution et la spatialisation 3D d'un grand nombre de canaux audio.

Le langage JavaScript ne s'exécute pas dans un processus temps réel et peut se voir refuser la priorité par d'autres processus systèmes en cours d'exécution. Certaines fonctionnalités peuvent rajouter des délais imprédictibles au niveau des processus JavaScript. Par ailleurs, de multiples instructions JavaScript peuvent être exécutées dans un même processus principal, tandis que d'autres processus en cours d'exécution (autres que JavaScript) tentent de l'utiliser au même moment.

Pour les développeurs, il fut très difficile de s'engager sur le maintien de latences fixes pour la totalité des process audio mis en œuvre, nul doute qu'elles augmentent avec le chaînage de différents modules, le tout restant dépendant de la puissante du système. Même si différentes horloges peuvent être interrogées (*Web Audio Clock*, *JavaScript Clock*), elles sont plus ou moins précises et leur contrôle est délicat. Il n'existe pas de mécanisme de compensation automatique des latences induites, comme le proposent les consoles de mixage et autres systèmes audionumériques professionnels.

…sur les conséquences pour le sous-projet :

L'API Web Audio reste une solution idéale pour rendre les services d'accessibilité audio présentés dans le cadre du projet media4Dplayer. Cette boîte à outils permet aussi de démontrer l'apport effectif de fonctionnalités audio avancées qui augmente l'expérience utilisateur de façon significative, et lui autorise une personnalisation quasi totale des réglages d'écoute. Devant les incertitudes avérées à opérer totalement en HTML5, telles que remontées par la présente étude, il a été décidé :

- de confier à l'IRCAM la totalité des développements des modules audio avancés (voir livrable 3.2.2), en utilisant expressément les nœuds natifs intégrés à l'API Web Audio pour les traitements les plus simples, et à limiter l'ajout de nouvelles fonctionnalité aux seuls modules qui le nécessitent obligatoirement ;
- de proposer une solution "native" (programmé en langage C/C++) pour quelques modules vraisemblablement plus complexes, trop gourmand en terme de process et de consommation de ressource en programmation purement JavaScript ;
- de repartir de la preuve de concept **FTVSync** en confiant au prestataire d'origine la mise à jour du démonstrateur iOS natif permettant de s'interfacer avec l'environnement media4Dplayer (accès à la liste des médias qui intègrent une audiodescription, aux ressources

additionnelles pour calibrer l'application second écran, compatibilité MPEG-DASH, opérations en mode fichier, resynchronisation récurrente). Cette dernière décision a un impact direct sur plusieurs livrables et restreint le périmètre du lot 3.4, sachant que le benchmarking des algorithmes de synchronisation se réduit à la version développée précédemment, et que le code source JavaScript prévu à l'origine se voit remplacé par une application iOS native.